

#01 criando o projeto com nest

Comando `nest new nomeDoProjeto`, remova tudo de teste e tbm do package.json de test

```
22   "rxjs": "^7.8.1"
23 },
24 "devDependencies": {
25   "@nestjs/cli": "^10.0.0",
26   "@nestjs/schematics": "^10.0.0",
27   "@nestjs/testing": "^10.0.0",
28   "@types/express": "^4.17.17",
29   "@types/jest": "^29.5.2",
30   "@types/node": "^20.3.1",
31   "source-map-support": "^0.5.21",
32   "ts-node": "^10.9.1",
33   "tsconfig-paths": "^4.2.0",
34   "typescript": "^5.1.3"
35 }
36 }
```

#02 Módulos, serviço e controllers

Controllers é a porta de entrada para nossa aplicação via http, é tudo que recebe requisição http.

Decorators é um função que adiciona comportamento em algo.

Módulo é o arquivo que junta tudo,.

Services pode ser qualquer coisa, tudo que não recebe requisição http é provider

#03 configurando ESLint e Prettier `pnpm i eslint @rocketseat/eslint-config -D` o comando é `pnpm run lint`

```
app.module.ts U  .eslintrc.json U X
.eslintrc.json > ...
1  {
2    "extends": "@rocketseat/eslint-config/node",
3    "rules": {
4      "no-useless-constructor": "off"
5    }
6  }

.eslintignore U X
.eslintignore
1  node_modules
2  dist
```

#04 setup Docker compose

```
docker-compose.yml
1  version: '3.8'
2
3  services:
4    postgres:
5      container_name: nest-clean-postgres
6      image: postgres
7      ports:
8        - 5432:5432
9      environment:
10       POSTGRES_USER: postgres
11       POSTGRES_PASSWORD: docker
12       POSTGRES_DB: nest-clean
13       PGDATA: /data/postgres
14      volumes:
15        - ./data/pg:/data/postgres
```

Rodar o comando `Docker-compose up -d`

#05 setup do prisma `pnpm prisma -D` e `pnpm i @prisma/client` depois rodar o comando `pnpm prisma init`

```
.gitignore
35 !.vscode/extensions.json
36
37 # Env
38 .env
39
40 # Docker
41 data
```

```
13 model User {
14   id      String @id @default(uuid())
15   name    String
16   email   String @unique
17   password String
18
19   questions Question[]
20
21   @@map("users")
22 }
23
24 model Question {
25   id      String @id @default(uuid())
26   title   String
27   slug    String @unique
28   content String
29   createdAt DateTime @default(now()) @map("created_at")
30   updatedAt DateTime? @updatedAt
31   authorId String @map("author_id")
32
33   author User @relation(fields: [authorId], references: [id])
34
35   @@map("questions")
36 }
```

rodar o comando para gerar as migrations `pnpm prisma migrate dev`

```
schema.prisma U  .env x docker-compose.yml U
1 DATABASE_URL="postgresql://postgres:docker@localhost:5432/nest-clean?schema=public"
```

Abrir no navegador `pnpm prisma studio`

#06 Criando serviço do prisma

```

1 import { Injectable, OnModuleDestroy, OnModuleInit } from '@nestjs/common'
2 import { PrismaClient } from '@prisma/client'
3
4 @Injectable()
5 export class PrismaService extends PrismaClient implements OnModuleInit, OnModuleDestroy {
6   public client: PrismaClient
7
8   constructor() {
9     super({
10       log: ['warn', 'error'] //faz um log do warn e error
11     }) //chama o construtor da classe
12   }
13
14   onModuleInit() { //chama quando for instanciado
15     return this.$connect() //conecta com prisma
16   }
17
18   onModuleDestroy() { //chama quando for destruido
19     return this.$disconnect() //desconecta do prisma caso caia
20   }
21 }

```

```

6 @Module({
7   controllers: [AppController],
8   providers: [AppService, PrismaService],
9 })
10 export class AppModule {}
11

```

```

1 import { Controller, Get, Post } from '@nestjs/common'
2 import { AppService } from '../app.service'
3 import { PrismaService } from '../prisma/prisma.service'
4
5 @Controller()
6 export class AppController {
7   constructor(
8     private appService: AppService,
9     private prisma: PrismaService,
10   ) {}
11
12   @Get()
13   getHello(): string {
14     return this.appService.getHello()
15   }
16
17   @Post('/hello')
18   async store() {
19     return await this.prisma.user.findMany()
20   }
21 }

```

#07 Controller de criação de conta

```

14 "skipLibCheck": true,
15 "strict": true,
16 "strictNullChecks": true,
17 "noImplicitAny": false,
18 "strictBindCallApply": false,
19 "forceConsistentCasingInFileNames": false,

```

```
src > controllers > create-account-controllers > CreateAccountController
1 import { Body, ConflictException, Controller, HttpCode, Post } from '@nestjs/common'
2 import { PrismaService } from '../prisma/prisma.service'
3
4 @Controller('/accounts')
5 export class CreateAccountController {
6   constructor(private prisma: PrismaService) {} //chama o construtor
7
8   @Post()
9   @HttpCode(201) //força o retorno 201
10  async handle(@Body() body: any) { //vem do corpo e salva na var body
11    const { name, email, password } = body //pega de dentro do body
12
13    const userWithSameEmail = await this.prisma.user.findUnique({
14      where: {
15        email
16      }
17    })
18
19    if (userWithSameEmail) {
20      throw new ConflictException("User with same e-mail address already exists.")
21    }
22
23    await this.prisma.user.create({
24      data: {
25        name, email, password
26      }
27    })
28  }
29 }
```

#08 gerando hash de senha pnpm i bcryptjs e o pnpm i @types/bcryptjs no controller de user

```
21 throw new ConflictException("User with same e-mail address already exists.")
22 }
23
24 const hashedPassword = await hash(password, 8)
25
26 await this.prisma.user.create({
27   data: {
28     name,
29     email,
30     password: hashedPassword
31   }
32 })
33 }
34 }
```

#09 criando pipe de validação do zod pnpm i zod, pipes são middlewares interceptadores

```
6 //cria o schema do zod
7 const createAccountBodySchema = z.object({
8   name: z.string(),
9   email: z.string().email(),
10  password: z.string(),
11 })
12
13 type CreateAccountBodySchema = z.infer<typeof createAccountBodySchema>
14
15 @Controller('/accounts')
16 export class CreateAccountController {
17   constructor(private prisma: PrismaService) {} //chama o construtor
18
19   @Post()
20   @HttpCode(201) //força o retorno 201
21   async handle(@Body() body: CreateAccountBodySchema) { //vem do corpo e salva na var body
22     const { name, email, password } = createAccountBodySchema.parse(body) //pega de dentro do body, validando com zod
23
24     const userWithSameEmail = await this.prisma.user.findUnique({
25       where: {
```

```
src > pipes > zod-validation-pipes > ...
1 import { PipeTransform, ArgumentMetadata, BadRequestException } from '@nestjs/common'
2 import { ZodSchema } from 'zod'
3
4 export class ZodValidationPipe implements PipeTransform {
5   constructor(private schema: ZodSchema) {}
6
7   transform(value: unknown) {
8     try {
9       const parsedValue = this.schema.parse(value);
10      return parsedValue;
11    } catch (error) {
12      throw new BadRequestException('Validation failed');
13    }
14  }
15 }
```

```

@Post()
@HttpCode(201) //força o retorno 201
@UsePipes(new ZodValidationPipe(createAccountBodySchema)) //usando o pipe do zod
async handle(@Body() body: CreateAccountBodySchema) { //vem do corpo e salva na var body
  const { name, email, password } = body //pega de dentro do body, validando com zod

```

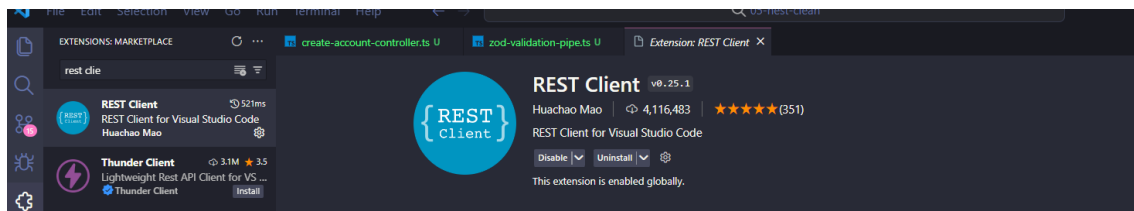
Instalar pnpm i zod-validation-error

```

src > pipes > zod-validation-pipe.ts > ...
1  import { PipeTransform, BadRequestException } from '@nestjs/common'
2  import { ZodError, ZodSchema } from 'zod'
3  import { fromZodError } from 'zod-validation-error'
4
5  export class ZodValidationPipe implements PipeTransform {
6    constructor(private schema: ZodSchema) {}
7
8    transform(value: unknown) {
9      try {
10         const parsedValue = this.schema.parse(value);
11         return parsedValue;
12       } catch (error) {
13         if (error instanceof ZodError) {
14           throw new BadRequestException({
15             message: 'Validation failed',
16             statusCode: 400,
17             errors: fromZodError(error)
18           }); //forma o error de forma mais visual
19         }
20       }
21       throw new BadRequestException('Validation failed');
22     }
23   }
24 }

```

#10 extensão rest cliente no vscode



Separar as requisições por

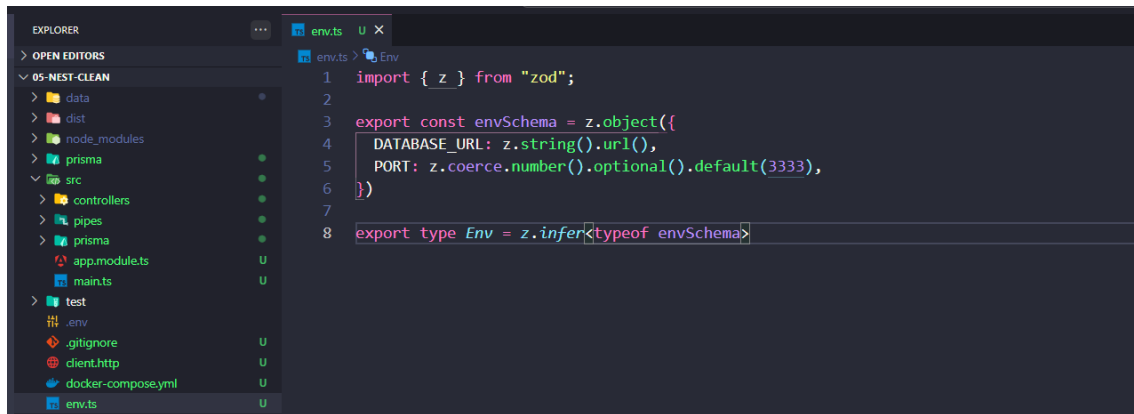
```

EXPLORER
  05-NEST-CLEAN
    > data
    > dist
    > node_modules
    > prisma
    > src
      > controllers
      > pipes
      > zod-validation-pipe.ts
    > prisma
    > app.module.ts
    > main.ts
    > test
    > .env
    > .gitignore
    > client.http
    > docker-compose.yml
    > nest-cli.json
    > package.json
    > pnpm-lock.yaml
    > tsconfig.build.json
    > tsconfig.json

client.http
  2 references
  1  @baseUrl = http://localhost:3333
  2
  3  # @name create_account
  4  Send Request
  5  POST {{baseUrl}}/accounts
  6  Content-Type: application/json
  7
  8  {
  9    "name": "Maria santos",
 10    "email": "email@gmail.com",
 11    "password": "12345"
 12  }
 13  ###
 14
 15  # @name authenticate
 16  Send Request
 17  POST {{baseUrl}}/sessions
 18  Content-Type: application/json
 19
 20  {
 21    "email": "email@gmail.com",
 22    "password": "12345"
 23  }

```

#11 usando configmodule no nest.js pnpm i @nestjs/config



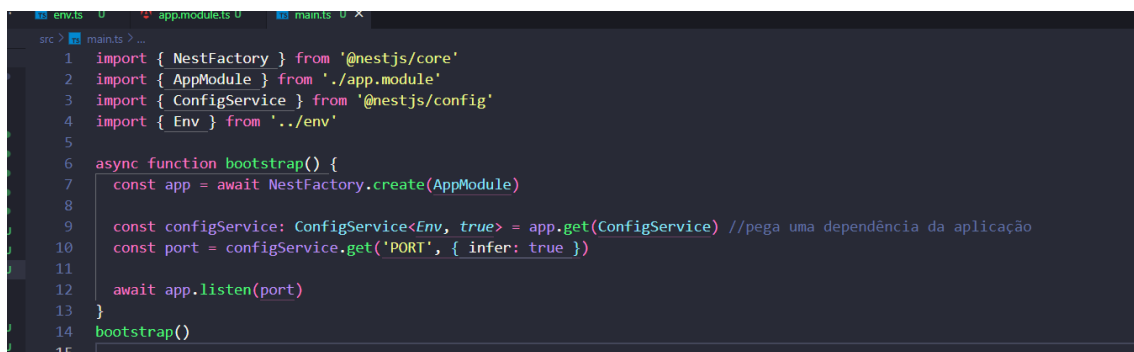
The Explorer sidebar on the left shows a project structure with folders like data, dist, node_modules, prisma, and src. The src folder is expanded, showing subfolders like controllers, pipes, prisma, app.module.ts, main.ts, and test. The main.ts file is selected and open in the editor. The code in main.ts defines an environment schema using zod and exports an Env type.

```
1 import { z } from "zod";
2
3 export const envSchema = z.object({
4   DATABASE_URL: z.string().url(),
5   PORT: z.coerce.number().optional().default(3333),
6 });
7
8 export type Env = z.infer<typeof envSchema>
```



The editor shows the app.module.ts file. It imports necessary modules from @nestjs/common, prisma, and the local controllers directory. It also imports ConfigModule from @nestjs/config and the Env type from ../env. The AppModule is decorated with @Module and configured with imports, controllers, and providers. Finally, the AppModule class is exported.

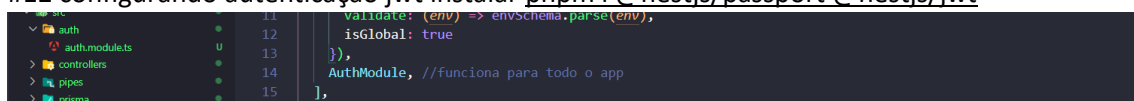
```
1 import { Module } from '@nestjs/common'
2 import { PrismaService } from '../prisma/prisma.service'
3 import { CreateAccountController } from '../controllers/create-account-controller'
4 import { ConfigModule } from '@nestjs/config'
5 import { envSchema } from '../env'
6
7 @Module({
8   imports: [ConfigModule.forRoot({
9     validate: (env) => envSchema.parse(env),
10     isGlobal: true
11   })],
12   controllers: [CreateAccountController],
13   providers: [PrismaService],
14 })
15 export class AppModule {}
16
```



The editor shows the main.ts file. It imports NestFactory from @nestjs/core, AppModule from ./app.module, ConfigService from @nestjs/config, and the Env type from ../env. The bootstrap function is defined as an async function that creates the app, gets the ConfigService, retrieves the port, and listens on it. The bootstrap function is then called.

```
1 import { NestFactory } from '@nestjs/core'
2 import { AppModule } from './app.module'
3 import { ConfigService } from '@nestjs/config'
4 import { Env } from '../env'
5
6 async function bootstrap() {
7   const app = await NestFactory.create(AppModule)
8
9   const configService: ConfigService<Env, true> = app.get(ConfigService) //pega uma dependência da aplicação
10   const port = configService.get('PORT', { infer: true })
11
12   await app.listen(port)
13 }
14 bootstrap()
15
```

#12 configurando autenticação jwt instalar pnpm i @nestjs/passport @nestjs/jwt



The Explorer sidebar shows the project structure. The src folder is expanded, showing subfolders like auth, controllers, pipes, and prisma. The auth folder is selected, and the auth.module.ts file is open in the editor. The code in auth.module.ts defines an environment schema using zod and exports an Env type.

```
11 validate: (env) => envSchema.parse(env),
12 isGlobal: true
13 }],
14 AuthModule, //funciona para todo o app
15 ],
```



The editor shows the env.ts file. It imports zod and defines an environment schema with DATABASE_URL, JWT_SECRET, and PORT. The schema is exported as envSchema, and the Env type is exported as z.infer<typeof envSchema>.

```
1 import { z } from "zod";
2
3 export const envSchema = z.object({
4   DATABASE_URL: z.string().url(),
5   JWT_SECRET: z.string(),
6   PORT: z.coerce.number().optional().default(3333),
7 });
8
```

```

1 import { Module } from '@nestjs/common'
2 import { PassportModule } from '@nestjs/passport'
3 import { JwtModule } from '@nestjs/jwt'
4 import { ConfigService } from '@nestjs/config'
5 import { Env } from '../env'
6
7 @Module({
8   imports: [
9     PassportModule,
10    JwtModule.registerAsync({
11      inject: [ConfigService], //lista de serviço injetado quando registro esse módulo
12      useFactory: (config: ConfigService<Env, true>) {
13        const secret = config.get('JWT_SECRET', { infer: true })
14      },
15      return: {
16        secret,
17      },
18    }),
19  ],
20 })
21 export class AuthModule {}

```

#13 gerando token jwt o comando para gerar é “openssl genpkey -algorithm RSA -out private_key.pem -pkeyopt rsa_keygen_bits:2048” e depois “openssl rsa -pubout -in private_key.pem -out public_key.pem”

<https://travistidwell.com/jsencrypt/demo/>

`certutil -encode private_key.pem private_key-base64.txt`

`certutil -encode public_key.pem public_key-base64.txt`

```

1 import { z } from "zod";
2
3 export const envSchema = z.object({
4   DATABASE_URL: z.string().url(),
5   JWT_PRIVATE_KEY: z.string(),
6   JWT_PUBLIC_KEY: z.string(),
7   PORT: z.coerce.number().optional().default(3333),
8 })
9
10 export type Env = z.infer<typeof envSchema>

```

```

@Module({
  imports: [
    PassportModule,
    JwtModule.registerAsync({
      inject: [ConfigService], //lista de serviço injetado quando registro esse módulo
      global: true,
      useFactory: (config: ConfigService<Env, true>) {
        const privateKey = config.get('JWT_PRIVATE_KEY', { infer: true })
        const publicKey = config.get('JWT_PUBLIC_KEY', { infer: true })

        return {
          signOptions: { algorithm: 'RS256' },
          privateKey: Buffer.from(privateKey, 'base64'),
          publicKey: Buffer.from(publicKey, 'base64'),
        }
      },
    }),
  ],
})
export class AuthModule {}

```

```

1 import { Controller, Post } from '@nestjs/common'
2 import { JwtService } from '@nestjs/jwt'
3
4 @Controller('/sessions')
5 export class AuthenticateController {
6   constructor(private jwt: JwtService) {}
7
8   @Post()
9   async handle() {
10     const token = this.jwt.sign({ sub: 'user-id' })
11     return token
12   }
13 }

```

```

16 ],
17 controllers: [CreateAccountController, AuthenticateController],
18 providers: [PrismaService],
19 }

```

#14 controller de autenticação

```

7
8 //cria o schema do zod
9 const authenticateBodySchema = z.object({
10   email: z.string().email(),
11   password: z.string(),
12 })
13
14 type AuthenticateBodySchema = z.infer<typeof authenticateBodySchema>
15
16 @Controller('/sessions')
17 export class AuthenticateController {
18   constructor(
19     private prisma: PrismaService,
20     private jwt: JwtService
21   ) {}
22
23   @Post()
24   @UsePipes(new ZodValidationPipe(authenticateBodySchema)) //usando o pipe do zod
25   async handle(@Body() body: AuthenticateBodySchema) {
26     const { email, password } = body
27
28     const user = await this.prisma.user.findUnique({
29       where: { email }
30     })
31
32     if (!user) {
33       throw new UnauthorizedException("User credentials do not match.")
34     }
35
36     const isValidPassword = await compare(password, user.password)
37
38     if (!isValidPassword) {
39       throw new UnauthorizedException("User credentials do not match.")
40     }
41
42     const accessToken = this.jwt.sign({ sub: user.id })
43
44     return {
45       access_token: accessToken
46     }
47   }
48 }


```

#15 protegendo rotas com guards instalar o `pnpm i passport-jwt` e `pnpm i @types/passport-jwt`

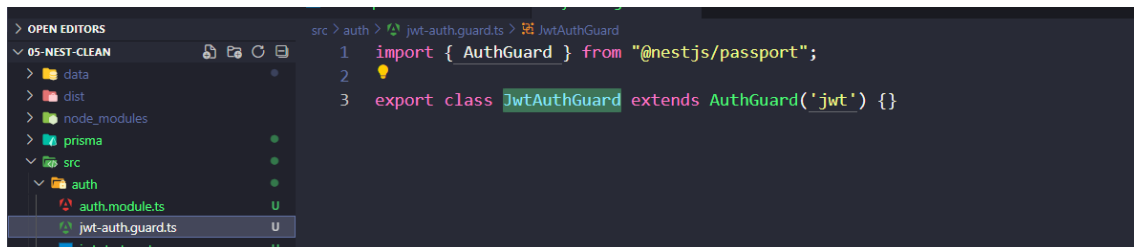
```

5 import { Env } from '../env'
6 import { JwtStrategy } from './jwt.strategy'
7
8 @Module({
9   imports: [
10     JwtModule.register({
11       secret: Env.JWT_SECRET,
12       signOptions: { expiresIn: '1h' }
13     })
14   ],
15   providers: [JwtStrategy]
16 })
17 export class AuthModule {}

```

```
1 import { ConfigService } from '@nestjs/config'
2 import { PassportStrategy } from '@nestjs/passport'
3 import { ExtractJwt, Strategy } from 'passport-jwt'
4 import { Env } from '../env'
5 import { z } from 'zod'
6 import { Injectable } from '@nestjs/common'
7
8 const tokenSchema = z.object({
9   sub: z.string().uuid()
10 })
11
12 type TokenSchema = z.infer<typeof tokenSchema>
13
14 @Injectable()
15 export class JwtStrategy extends PassportStrategy(Strategy) {
16   constructor(config: ConfigService<Env, true>) {
17     const publicKey = config.get('JWT_PUBLIC_KEY', { infer: true })
18
19     super({
20       jwtFromRequest: ExtractJwt.fromAuthHeaderAsBearerToken(), //pega o token do header
21       secretOrKey: Buffer.from(publicKey, 'base64'),
22       algorithms: ['RS256']
23     }) //chama o construtor da classe strategypassport
24
25     async validate(payload: TokenSchema) {
26       return tokenSchema.parse(payload)
27     }
28   }
29 }
```

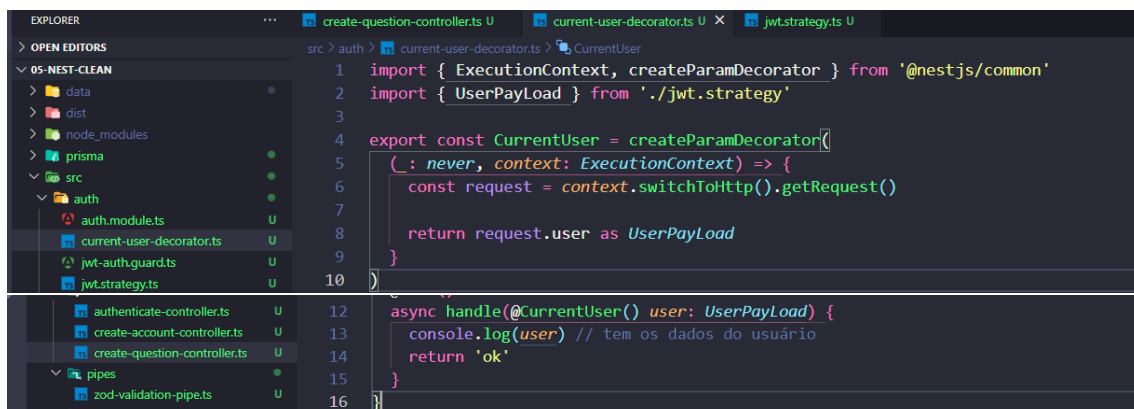


```
1 import { AuthGuard } from '@nestjs/passport';
2
3 export class JwtAuthGuard extends AuthGuard('jwt') {}
```



```
1 import { Controller, Post, UseGuards, UsePipes } from '@nestjs/common'
2 import { JwtAuthGuard } from '../auth/jwt-auth.guard'
3
4 @Controller('/questions')
5 @UseGuards(JwtAuthGuard)
6 export class CreateQuestionController {
7   constructor() {}
8
9   @Post()
10   async handle() {
11     return 'ok'
12   }
13 }
```

#16 criando decorator de autenticação



```
1 import { ExecutionContext, createParamDecorator } from '@nestjs/common'
2 import { UserPayload } from '../jwt.strategy'
3
4 export const currentUser = createParamDecorator(
5   ( : never, context: ExecutionContext ) => {
6     const request = context.switchToHttp().getRequest()
7
8     return request.user as UserPayload
9   }
10 )
11
12 async handle(@currentUser() user: UserPayload) {
13   console.log(user) // tem os dados do usuário
14   return 'ok'
15 }
16 }
```

#17 controller de criação de pergunta

```
create-question-controller.ts U x create-account-controller.ts U
src > controllers > create-question-controller.ts > CreateQuestionController > handle
9   const createQuestionBodySchema = z.object({
10     title: z.string(),
11     content: z.string(),
12   })
13
14   type CreateQuestionBodySchema = z.infer<typeof createQuestionBodySchema>
15
16   @Controller('/questions')
17   @UseGuards(JwtAuthGuard)
18   export class CreateQuestionController {
19     constructor(
20       private prisma: PrismaService
21     ) {}
22
23     @Post()
24     async handle(
25       @Body(new ZodValidationPipe(createQuestionBodySchema)) body: CreateQuestionBodySchema,
26       @CurrentUser() user: UserPayload
27     ) {
28       console.log(user) // tem os dados do usuário
29       const { content, title } = body
30       const { sub: userId } = user
31
32       const slug = this.convertToSlug(title)
33
34       await this.prisma.question.create({
35         data: {
36           authorId: userId,
37           title,
38           content,
39           slug
40         }
41       })
42     }
43
44     private convertToSlug(title: string): string {
45       return title
46         .toLowerCase()
47         .normalize('NFD')
48         .replace(/[\u0300-\u036f]/g, '')
49         .replace(/[^w\s-]/g, '')
50         .replace(/s+/g, '-')
51     }
52   }
```

#18 controller de listagem de pergunta

#19 configurando vitest com swc

#20 banco de dados isolando nos test

#21 testes e2e de usuário

#22 testes e2e de perguntas